DATA MODEL

A data model logically represents data objects, how they relate, and the rules that govern them. Data models are like blueprints for building efficient databases. A data model provides a way to structure complex information, for example, customer records or product inventory, into logical arrangements that make it easier for businesses to retrieve and manage information efficiently.

For example, a data model might relate an individual customer's profile to their purchase history to quickly retrieve it within a data query. Data models define relationships and constraints that limit how data can be correlated together while also acknowledging the types of data associated with each point of contact.

Good constraint relationships minimize data duplication and eliminate inconsistency or poorly functioning systems that leverage data inappropriately, ensuring that errors do not occur.

Understanding data models is essential for designing and implementing databases because they relate directly to system performance and scalability. A well-designed data model will improve query performance, preserve data integrity, and position the organization to grow its user base, geographic reach, or extensibility.

Choosing the right model can significantly reduce processing time on a movement for a retail company with millions of transactions in a single day.

1. Conceptual Model in DBMS

The conceptual data model in DBMS is a high-level interface. This model will not explain the technical details of the data. This will only help the user to know and understand the concept. The concepts were explained in attributes and relationships with the help of an Entity-Relationship (ER) diagram. The ER diagram will be helpful to let the stakeholders know the hierarchy and structure of the data.

Example: If the database wants to store the banking data, the conceptual model will create an Entity Relationship diagram (ER) with the manager, transactions, account, and user ID. This entity diagram will allow the stakeholders and users to understand the structure of the database.

2. Logical Model in DBMS

The Logical data model in DBMS helps decide the elements that are present in the database. This model will decide the primary keys, foreign keys, attributes, and the relationship between them. It will ensure normalization and data integrity. This model will represent data in a table format, but neither of the entities will depend on a specific database.

Example: In a banking system, the user will have their own user ID and transaction ID, which will be connected to their loans in a particular bank.

3. Physical Model in DBMS

The physical data model in DBMS is the actual implementation of the database. It will show the user how the data is stored and managed in a database. The physical model will represent and decide the details of table structures, where they will be stored, the constraints that can be used, the table indexes, the partitions of tables or data, and the type of data that can be used to make the database more efficient. This model will transform the logical design into a working schema with the help of the SQL language. SQL will optimize the query performance by indexing and storing data efficiently.

Example: MySQL is very efficient for implementing the Physical Model.

Types of Data Models in DBMS

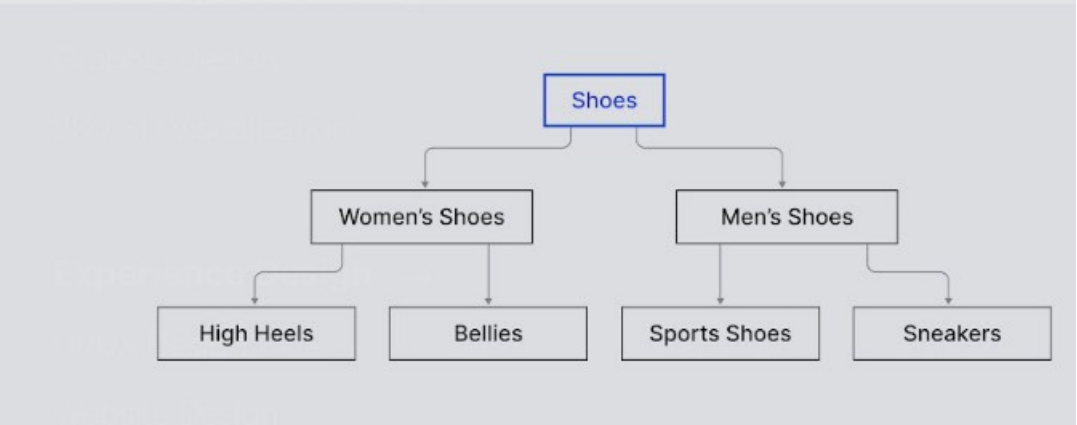
Let’s understand different types of data models in DBMS, characteristics, and practical applications, including type and feature comparison tables.

1. Hierarchical Model

The Hierarchical Model organizes data in a tree structure with a parent-child relationship. Each parent record can have multiple child records, but each child is associated with only one parent. Hierarchical models are best suited to applications that have simple hierarchies, such as an organization chart and file systems. However, they struggle to handle complex relationships due to the limitations of their structure.

Examples: IBM Information Management System (IMS),

Windows Registry, XML Data Storage, File System Directories.



Advantages of the Hierarchical Model

- **Simplicity:** The tree-like structure of the hierarchical model is intuitive and easy to understand, which simplifies the design and navigation of databases. This structure clearly defines parent-child relationships, making the path to retrieve data straightforward.
- **Data Integrity:** Due to its rigid structure, the hierarchical model inherently maintains data integrity. Each child record has only one parent, which helps prevent redundancy and preserves the consistency of data across the database.
- **Efficient Data Retrieval:** The model allows for fast and efficient retrieval of data. Since relationships are predefined, navigating through the database to find a specific record is quicker compared to more complex models where paths are not as clearly defined.
- **Security:** Hierarchical databases can provide enhanced security. Access to data can be controlled by restricting users to certain segments of the tree, which makes it easier to implement security policies based on data hierarchy.
- **Performance:** For applications where relationships are fixed and transactions require high throughput, such as in banking or airline reservation systems, the hierarchical model offers excellent performance and speed.

Limitations of the Hierarchical Model

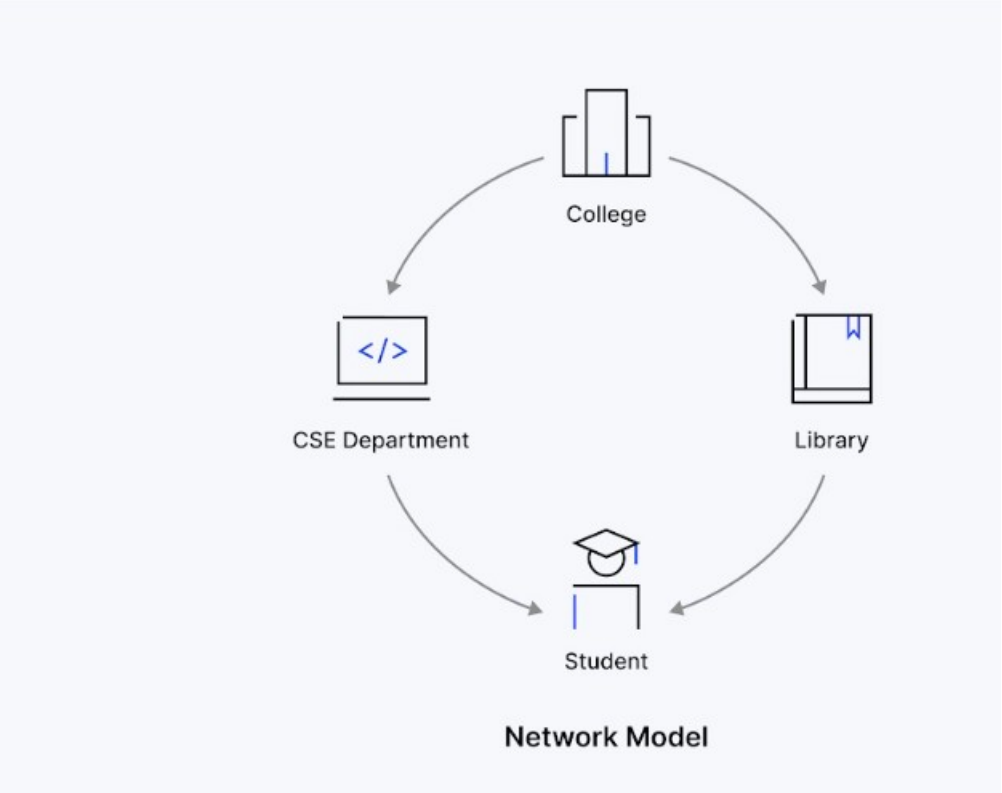
- **Structural Rigidity:** The biggest limitation of the hierarchical model is its structural rigidity. Once the database is designed, making changes to its structure, such as adding or modifying the relationships between records, can be complex and labor-intensive.

- **Lack of Flexibility:** This model does not easily accommodate many-to-many relationships between records, which are common in more complex business applications. This limitation can lead to redundancy and inefficiencies when trying to model more intricate relationships.
- **Data Redundancy:** In cases where a child has more than one parent, the hierarchical model can lead to data duplication because each child-parent relationship needs to be stored separately. This redundancy can consume additional storage and complicate data management.
- **Complex Implementation:** Implementing a hierarchical database can be more complex than more modern relational databases, particularly when dealing with large sets of data that have diverse and intricate relationships.
- **Query Limitations:** The hierarchical model typically requires a specific type of query language that might not be as rich or flexible as SQL, used in relational databases. This can limit the types of queries that can be executed, affecting the ease and depth of data analysis.

2. Network Model

The Network Model builds upon the Hierarchical Model by enabling a child record to have more than one parent, leading to a graph-like structure. It offers greater flexibility to support more complex relationships, such as supply chain networks. While the Network Model is efficient for certain use cases, it is difficult to maintain due to its complexity.

Examples: IDMS (Integrated Database Management System), Integrated Data Store (IDS), Raima Database Manager, Turboimage, CODASYL DBMS.



Advantages

- This model is more flexible than the hierarchical model but more complex to design.
- It is capable of handling multiple types of relationships which can help in modeling real-life applications.
- This network does not allow a member to exist without an owner which leads to the concept of Data integrity.
- Like a hierarchical model, this model also does not have any database standard.

Disadvantages

- The design or the structure of this model is not user-friendly and does not have any scope of automated query optimization.
- Once the the database of the model is formed it is difficult to change.
- There's an existence of operational anomalies as there is a use of pointers for navigation which further leads to complex implementation.

3. Entity-Relationship Model

The Entity-Relationship (ER) Model describes objects (entities) and the relationships among those entities for conceptual database design. The model is expressed using ER diagrams,

which define entities (e.g., customers), attributes (e.g., name, ID), and relationships (e.g., purchases). The Entity-Relationship model is most commonly used in the early stages of a database design for its simplicity.

Examples: MySQL Workbench, Oracle SQL Developer Data Modeler, Microsoft Visio, IBM InfoSphere Data Architect, Lucidchart, ER/Studio.

4. Relational Model

The Relational Model is the most widely used data model in DBMS. It organizes data as tables (called relations) consisting of rows and columns. Here, the tables representing entities and relationships are established using keys (e.g., primary keys and foreign keys). Its ease of use and flexibility make it a good fit for financial and customer relationship management (CRM) platforms. According to MMR, the RDBMS market is expected to reach USD 171.93 billion by 2030.

Examples: MySQL, PostgreSQL, Oracle Database, Microsoft SQL Server, SQLite, MariaDB

5. Object-Oriented Model

The object-oriented model incorporates object-oriented programming (OOP) concepts, such as objects, classes, and inheritance, into database systems. It stores both data and the operations (e.g., methods). This model is well-suited for environments like multimedia systems or CAD systems. While the object-oriented model handles complex data effectively, skill sets may span a specialized area.

Examples: db4o, ObjectDB, ObjectStore, Versant Object Database, GemStone/S, InterSystems Caché.

Advantages of Object Oriented Data Model :

- Codes can be reused due to inheritance.
- Easily understandable.
- Cost of maintenance can reduced due to reusability of attributes and functions because of inheritance.

Disadvantages of Object Oriented Data Model :

- It is not properly developed so not accepted by users easily.

6. NoSQL Model

The NoSQL Model encompasses different non-relational models designed to be highly scalable and flexible (unformatted or semi-structured data). The NoSQL Model includes types like Document, Key-Value, Column-Family, and Graph, each designed to meet specific needs. These databases have gained popularity because they're often used for big data and real-time applications. According to Gartner, NoSQL database solutions such as MongoDB, Amazon DynamoDB, Microsoft Azure Cosmos DB, and more have high ratings and good reviews.

Examples: MongoDB, Cassandra, Redis, Couchbase, DynamoDB, Neo4j, HBase, Amazon DocumentDB.

7. Graph Model

The Graph Model organizes data as nodes (entities) and edges (relationships). Ideal for highly interconnected data like social networks and recommendation engines, the Graph Model is optimal for analysing complex relationships (e.g., determining the shortest path between users) but can also be very costly to process for large sets of data.

Examples: Neo4j, Amazon Neptune, ArangoDB, OrientDB, TigerGraph, Microsoft Azure Cosmos DB (Gremlin API), JanusGraph.

8. Document Model

The Document Model, a subtype of NoSQL, stores data in JSON format (each document has key-value pairs). Document databases work well for content management and e-commerce since they flexibly accommodate semi-structured data. While

development is easier with a schema-less approach, querying can be slightly more complex.

Examples: MongoDB, CouchDB, Amazon DocumentDB, RavenDB, ArangoDB, MarkLogic, BaseX.

9. Key-Value Model

The Key-Value Model, another type of NoSQL, stores data as key-value pairs, just like in a dictionary. It's great for things like caching or managing user sessions. Unfortunately, the key-value model lacks the functionality to create complex queries based on relationships.

Examples: Redis, Amazon DynamoDB, Riak KV, Berkeley DB, Voldemort, Aerospike, LevelDB.

10. Column-Family Model

The Column-Family Model, another subtype of NoSQL, organizes data into column families. This model allows data to be stored and read efficiently for large-scale analytics like data warehouses. The Column-family Model is found in systems like Apache Cassandra but requires careful logical schema design to extract performance.

Examples: Apache Cassandra, HBase, ScyllaDB, Amazon Keyspaces, Hypertable.

11. Object-Relational Model

The Object-Relational Model merges the features of relational and object-oriented models. It stores objects and complex data types (e.g., multimedia) in a relational database. Databases that implement this model include PostgreSQL. While object-relational databases provide more flexibility than purely relational data systems, they also introduce additional complexity.

Examples: PostgreSQL, Oracle Database, IBM Db2, Informix, Microsoft SQL Server (with limited object-relational features).

Comparison of Data Models

Now, let’s compare data models in a DBMS and identify key considerations for selecting a data model.

Data Model	**Features**	**Benefits**	**Limitations**
Hierarchical	Tree-like structure, parent-child links	Fast for hierarchical queries, simple	Rigid, struggles with complex relations
Network	Graph-like, multi-parent link	Handles complex relationships	Complex to design and maintain
Entity-Relationship	Entities, attributes, relationships	Clear for conceptual design	Not used for physical storage
Relational	Tables with rows, columns, and keys	Flexible, widely supported, easy to query	Slower for very large datasets
Object-Oriented	Objects, classes, inheritance	Handles complex data (e.g., multimedia)	Requires specialised skills
NoSQL - Graph	Nodes and edges for relationships	Fast for interconnected data queries	Resource-intensive for large graphs
NoSQL - Document	JSON/BSON documents, schema-less	Flexible for semi-structured data	Complex queries can be challenging
NoSQL - Key-Value	Simple key-value pairs	High performance for simple lookups	Limited to complex relationships
NoSQL - Column-Family	Column-based storage	Scalable for analytics, large datasets	Requires careful design for efficiency
Object-Relational	Combines relational and object features	Supports complex data in relational format	Increased complexity in implementation

Considerations for Selecting a Data Model

Selecting the appropriate data model depends on the specific needs of the relevant application. Here are some factors to consider:

Data structure and complexity: It is essential to understand whether the data being recorded is structured (e.g., records of financial transactions), semi-structured (e.g., JSON documents), or unstructured (e.g., social media posts). If the data is structured, relational models may be most applicable. If some data is semi-structured or has interconnections, then NoSQL models (e.g., document, graph) may be more appropriate.

Scalability requirements: Many applications or systems generate data at a rapid pace, and thus, they need to consider NoSQL models, particularly Column-Family, which support big data platforms. Relational models must be highly optimised to support such a large data volume.

Query performance: Graph models will be well-suited if the application requires fast and complex data querying (e.g., analysing social networks). Key-value models would be superior for simple key lookups.

Development and Maintenance: Because relational models have many tools and practitioners, development and maintenance processes tend to be easier than if you used an object-oriented or network model. The network model may require specialised skills or training that can increase the cost of the model.

Use Case Specificity: Match the model properties with the application's purpose. For instance, a relational model is the best choice for a retail business using a DBMS to track inventory. Pick a document or database model for any service that lets users save and manage all their preferences for various media.

Future Flexibility: If your application needs change, you want an adjustable model. NoSQL application models are schema-less and can provide future flexibility to your organization, while many relational models provide stability and a selective structure.

By considering these points, we hope you can identify a data model that supports your organization's performance and operational goals. You can also check out the 2024 Stack Overflow Developer Survey for insights into preferred databases here.

Suppose organizations fully understand the advantages and trade-offs between different data models in a DBMS. In that case, they are in a perfect position to build a database that is efficient, scalable, and suited to their needs.

Advantages of Data Models

Below, we summarize the benefits of using data models in DBMS to help clarify their importance in managing your organization’s database system.

Reduced Complexity: Reduced complexity results from using data models to logically structure complex datasets for storing, accessing, and managing information. For example, a retailer using a Relational Model can define tables for customer data and orders, minimizing confusion and keeping application functions less complicated.

Data Integrity: Data integrity is supported by data models through relationship representations and constraints, such as foreign keys in a Relational Model, ensuring consistency, reducing errors, and preserving data reliability for decision-making.

Improved query efficiency: Query efficiency improves from well-designed data models that enable faster queries through less bulky data access, proper use of system resources, and models like the Graph Model that excel at querying connected data.

Scalability and flexibility: Data models like the flexible Document Model and the structured Relational Model help systems grow and stay effective. The Document Model works well for changing data, while the Relational Model is great for keeping data organized and consistent.

Ease of development and maintenance: Development and

maintenance become easy through data models that simplify database design and implementation. Entity-Relationship Models can be used for planning. Relational Models with standardised tools like SQL can reduce development time and support maintenance.

Support for varied use cases: Data models in a DBMS support different use cases. The Object-Oriented Model supports complex data types, and the Key-Value Model supports high-performance caching, allowing businesses to choose models that match their specific requirements.

Improved security and access control: Security improvements and access control are provided by data models. Data models use features like access control and stored procedures, with the relational model allowing user-level access to critical data while preventing unauthorised access.

Disadvantages of Data Models

Let’s also examine the potential drawbacks associated with data models in DBMS.

High startup costs: DBMS implementations involve expensive technologies and skilled personnel, with firms potentially spending the majority of their IT budget on database optimisation, making it challenging for smaller businesses to deploy data models.

Performance trade-offs: Performance limitations occur as some models prioritise specific functionalities; for example, the Relational Model is great for structured data but may not handle large unstructured datasets as well as NoSQL models. Graph Models can be resource-heavy and lead to slower query speeds with larger datasets.

Implementation complexity: Implementation of these models can take a substantial level of expertise to use effectively (e.g., developing a Network Model to manage complex relationships or a Column-Family Model for big data analytics with little capacity of experience by the team can be problematic, inefficient, and can lead to errors).